消息队列 CKafka 版

故障处理

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

故障处理
Topic 相关故障
Topic 创建失败

最近更新时间：2024-10-14 14:29:51

问题概述

Topic 创建失败。

可能原因

- 已创建的所有 Topic 的分区数之和达到实例规格的分区数上限。
- 在删除 Topic 期间创建同名 Topic。
- Topic 已经存在集群当中。

解决方法

- **已创建的所有 Topic 的分区数之和达到实例规格的分区数上限**
  建议对 Kafka 实例扩容，或者删除不需要的 Topic。

- **在删除 Topic 期间创建同名 Topic**
  Topic 删除是异步操作，下发删除指令后，系统会异步的删除该 Topic 的元数据。若在此期间创建同名 Topic，系统会提示 Topic 已经存在，届时请您稍后重试。这里需要等待1分钟左右，再进行重建操作。
- **Topic 已经存在集群当中**
  Topic 已经存在集群当中，创建重名的 Topic 就会提示 Topic 已经存在，建议重新设置 Topic 名称。

Topic 无监控数据

最近更新时间：2024-10-11 10:15:07

问题概述

Topic 无监控数据。

可能原因

- 客户端可能没生产消费。
- 监控采集系统存在故障。

解决方法

- **客户端可能没生产消费**
  需确认客户端是否有生产消费的行为，可以使用原生的生产消费命令进行测试，再查看监控数据。具体操作参见 运行 Kafka 客户端 。
- **监控采集系统存在故障**
  如果监控采集系统存在故障，请 提交工单 处理。

Topic 配置 ACL 策略后导致其他产品联动能力失效

最近更新时间：2024-10-11 10:15:22

问题概述

为 Topic 配置 ACL 策略后导致其他产品联动能力失效。

可能原因

默认情况下，Topic 是没有配置 ACL 的，在同一个 VPC 内可以自由访问该 Topic。如果需要在 VPC 内做权限控制，可以参见 配置 ACL 策略 配置 ACL。
当为 Topic 添加一条 ACL 策略后，该策略会阻止其他所有不符合条件的请求访问 Topic，包括其他云产品联动 CKafka 的调用（例如：日志服务 CLS 的日志投递、云函数 SCF 消息转储、大数据 EMR 组件的消费等）。
从业务的角度来看，一旦业务为了保证设置了 ACL 后，就是不想要有不符合要求的客户端访问 Kafka 的数据，所以这个被拒绝也是合理的。

解决方法

在为某个 Topic 增加了一条 ACL 策略前，一定要通过业务信息或者控制台的监控信息判断 Topic 是否在其他场景中使用，否则很可能导致其他联动功能出现问题。
针对此类情况，如果您一定需要做 ACL 策略管理，建议您生产消息到一个新的 Topic 来做权限分配，不要复用原有 Topic。

分区的监控查看消息堆积数量一直存在

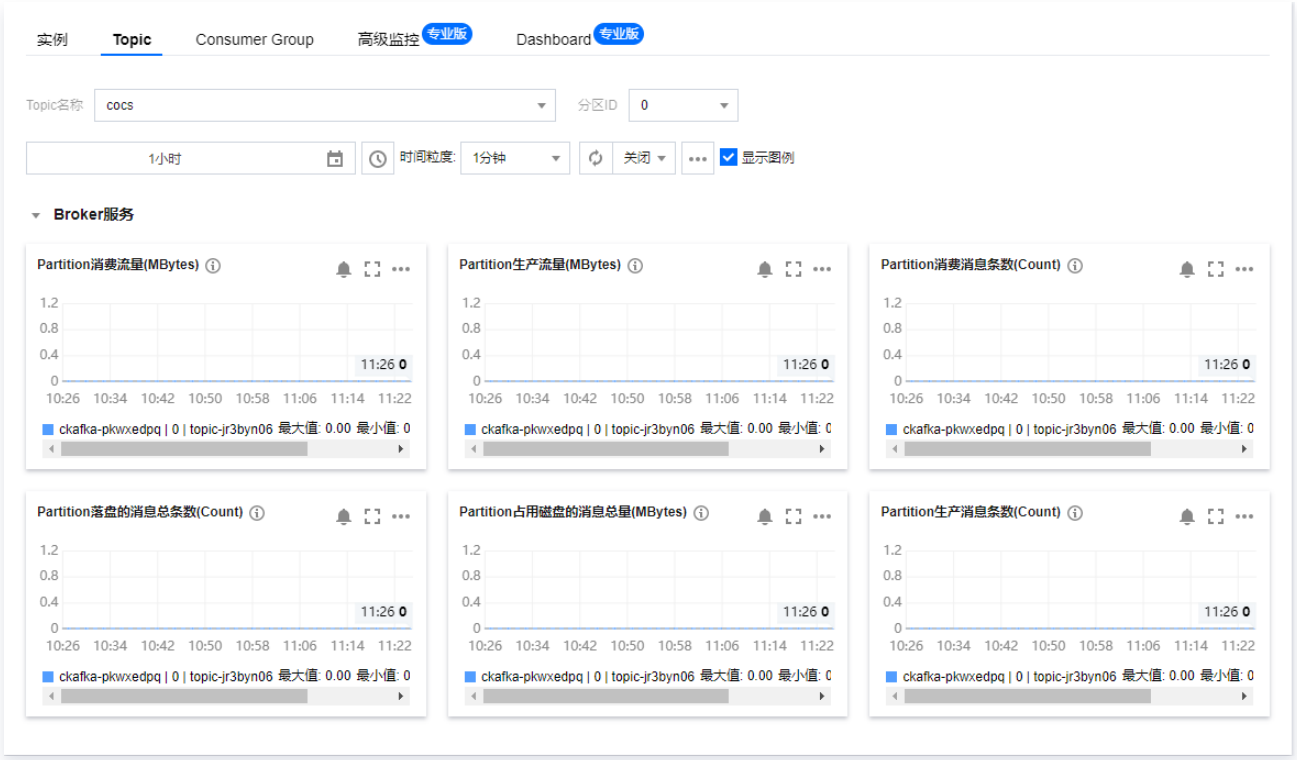
最近更新时间：2024-10-11 14:31:31

问题概述

某些分区的监控查看消息堆积数量一直存在。

可能原因

- 消费者没有对该分区进行消费，分区一直有生产消息行为，导致堆积数据一直存在。



- 消费者组中的消费者数量不足或者消费的速度过慢导致数据有堆积。
- 消费端程序存在 bug，可能存在不消费某些分区的情况。

解决方法

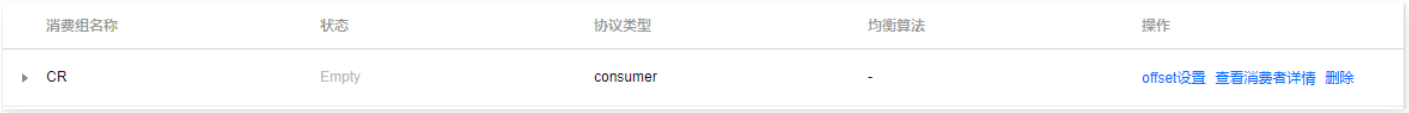
- **消费者没有对该分区进行消费，分区一直有生产消息行为，导致堆积数据一直存在**
  如果遇到不消费某些分区的情况，可以通过重启客户端的形式来尝试解决问题。如果重启可以解决问题，则先排查一下客户端日志，也可以 提交工单 请求协助。
- **消费者组中的消费者数量不足或者消费的速度过慢导致数据有堆积**
  可以尝试加大消费者的数量，以提高消费速度。
- **消费端程序存在 bug，可能存在不消费某些分区的情况**
  此时需要先排查消费端的本地日志，看是否有异常信息。

Consumer Group 相关故障
Consumer Group 列表详情缺失

最近更新时间：2024-10-10 14:29:51

现象概述

CKafka 的消费组列表有消费组名称，点开详情，却没有消费详情。例如：下图的消费组 CR 没有展示详情。



可能原因

Kafka 的数据消费有两种模式，消费组模式和自定义分区消费模式。

- 当使用消费组模式消费，客户端会通过消费协调者进行协调消费，在消费数据完成后，会往服务端提交 offset 的存储请求。则此时服务端会存储消费的 Topic、分区进度、客户端等信息。
- 当使用自定义分区消费的模式，则客户端不会自动往服务端提交 offset 存储请求，则此时如果客户端没有主动发起提交 offset 请求，则服务端是看不到消费的相关信息的。
- 当 Topic 设置了 ACL 以后，某些实例可能会出现无法看到消费者组的详情，如果出现无法看详情，请先检查是否有 ACL，如果有，则需要您 提交工单 进行处理。

解决方法

1. 查看实例的消费组列表。

```
]$ bin/kafka-consumer-groups.sh --bootstrap-server 9.146.153.249:9092
--list
CR
```

   可以看到当前的所有消费组名称。
2. 查看实例特定的消费组详情。

```
]$ bin/kafka-consumer-groups.sh --bootstrap-server 9.146.153.249:9092
--describe --group CR
Note: This will not show information about old Zookeeper-based
consumers.
```

   会发现该消费组并没有详情。这表示消费者客户端没有使用 consumerGroup 机制去消费数据。即客户端没有往服务端提交消费详情，服务端没有存储消费数据，则不会正常显示。

3. 定位是否是服务端的问题.

通过原生自带的消费组命令，指定消费组名称 test1 进行消费，如下所示:

```
]$ bin/kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092 --
from-beginning --topic test --group test1
```

则在控制台能正常显示的消费组，通过 `--describe` 命令是可以看到详情的，如下所示：

```
$ bin/kafka-consumer-groups.sh --bootstrap-server 127.0.0.1:9092 --describe --group test

GROUP     TOPIC    PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG   CONSUMER-ID                                                                    HOST           CLIENT-ID
test      test     0          -               67              -     sarama/127.0.0.12020-06-11 23:33:03:110-a113e7d1-e36e-44f5-867c-f3e2fa2a57ab  /127.0.0.1     sarama
```

Consumer Group 持续出现 PreparingRebalance 状态

最近更新时间：2024-10-10 09:38:32

问题概述

Consumer Group 持续出现 PreparingRebalance 的状态。

可能原因

1. 有新的消费者加入消费者组。
2. 当运行的消费者停止运行，离开消费者组。常见的情况如消费者重启，消费者应用崩溃，消费者进程上报的心跳超时等（详情参见 CKafka 常用参数配置指南 ）。
3. 分区数变动的时候（增加或删除）。

解决方法

1和3两种情况无法避免 rebalance。正常情况，rebalance 在30s内能完成。如果出现长期的 rebalance，需要 提交工单 进行处理。
如果是由于心跳超时或者两次 poll 时间间隔太大导致的问题，您可以调整如下参数：

```
# 使用 Kafka 消费分组机制时，消费者超时时间。当 Broker 在该时间内没有收到消费者的心跳时，认为该消费者故障失败，Broker 发起重新 Rebalance 过程。目前该值的配置必须在 Broker 配置group.min.session.timeout.ms=6000和 group.max.session.timeout.ms=300000 之间
session.timeout.ms=10000

# 使用 Kafka 消费分组机制时，消费者发送心跳的间隔。这个值必须小于 session.timeout.ms，一般小于它的三分之一
heartbeat.interval.ms=3000

# 使用 Kafka 消费分组机制时，再次调用 poll 允许的最大间隔。如果在该时间内没有再次调用 poll，则认为该消费者已经失败，Broker 会重新发起 Rebalance 把分配给它的 partition 分配给其他消费者
max.poll.interval.ms=300000
```

如果出现一个消费者订阅了很多的 Topic，您可以尝试减少 Group 订阅 Topic 的数量。

客户端相关故障
客户端常见报错与解决方案

最近更新时间：2024-10-11 16:38:16

客户端配置或服务异常

以下异常属于客户端配置或服务异常，客户端不会自动重试。

异常	描述	分析与说明
UnknownServerException	服务器处理请求发生未知错误。	老版本流控会返回这个错误；新版本则可能是服务器出现 BUG 导致。
RecordTooLargeException	消息太大。	目前配置 message.max.bytes=1000012。
InvalidRequiredAcksException	生产者配置的 acks 参数不合法。	-
InconsistentGroupProtocolException	Group 的协议不一致。	检查 Consumer 和 Connector 是否配置了相同的 group.id，这两者使用的不同的协议，不能加入相同的组。
InvalidGroupIdException	Consumer Group ID不合法。	建议使用 [a-zA-Z0-9._-] 这些字符，长度不超过 128。
InvalidTopicException	Topic 不合法。	开启自动创建 Topic 选项后，客户端使用的 Topic 不合法会返回这个异常。检查 Topic 是否使用了不合法的字符或长度是否超过限制。
InvalidSessionTimeoutException	消费者配置的 session.timeout.ms 不合法。	目前服务器端允许的最小值为：group.min.session.timeout.ms=6000，最大值为：group.max.session.timeout.ms=300000。
InvalidCommitOffsetSizeException	提交 Offset 信息太大超过最大消息大小，无法写入 __consumer_offsets。	目前配置 message.max.bytes=1000012。
OffsetMetadataTooLarge	Offset 提交请求包含的 Metadata 太大。	服务器配置的 offset.metadata.max.bytes=4096。

UnsupportedVersionException	Broker 不支持该版本的请求。	建议使用 0.10.2.x 版本的客户端。

程序正常运行时的短暂异常

以下异常在程序正常运行过程中可能会短暂出现，客户端会自动重试。 持续出现则服务不正常。

异常	描述	分析与说明
TimeoutException	请求超时。	首次连接报请求超时，先检查地址是否正确，telnet 确定网络能够联通。程序运行中偶尔抛出此异常可能属于网络抖动。
CorruptRecordException	消息不合法。	可能 CRC 检查不通过，数据大小不合法。此外，如果压缩方式使用 **GZip** 或 0.9 以下版本 **使用压缩** 也会导致这个错误。
UnknownTopicOrPartitionException	Topic 或 Partition 不存在。	到控制台检查是否已经创建对应的 Topic。注意：客户端通过 TopicName生产消费，而不是TopicId。此外，客户端没有权限访问 Topic 时也会报 Topic 不存在。
LeaderNotAvailableException	Partition 没有 Leader。	当 Topic 刚创建时服务器还未选出合适的 Leader，此时会返回此错误给客户端，客户端会自动重试获取 Leader信息。**旧版本才会有这个异常，0.10.2.1 已经去掉**。
NotLeaderForPartitionException	Partition 的 Leader 不可用。	由于客户端会缓存 Topic 的 Metadata，所以当 Partition 的 Leader 切换 时，生产或消费请求可能仍然发送到旧 Leader 上，此时会返回此错误给客户端，客户端会自动更新 Metadata 信息。
NetworkException	客户端连接被服务器端关闭。	网络异常或连接数超过限制。
NotEnoughReplicasException	ISR 数量不够。	在写入数据时 Partition 的 ISR 数量小于 Topic 配置的 min.insync.replicas，可能由于 ISR 抖动导致。
NotEnoughReplicasAfterAppendException	数据写入 Broker 本地后，发生 ISR 抖动导致无法满足	–

	min.insync.replicas。	
BrokerNotAvailableError	未找到该分区的 Leader。	由于客户端会缓存 Topic 的 Metadata，所以当 Partition 的 Leader 切换 时，生产或消费请求可能仍然发送到旧 Leader 上，此时会返回该错误给客户端。客户端会自动更新 Metadata 信息，在 Leader 切换后，新生产的请求发送到老的 Leader 报错应该后会自动调整到新的 Leader 上，理论上不会影响数据写入消费的完整性。
NotLeaderForPartitionError	未找到该分区的 Leader。	由于客户端会缓存 Topic 的 Metadata，所以当 Partition 的 Leader 切换 时，生产或消费请求可能仍然发送到旧 Leader 上，此时会返回此错误给客户端，客户端会自动更新 Metadata 信息，在 Leader 切换后，新生产的请求发送到老的 Leader 报错应该后会自动调整到新的 Leader 上，理论上不会影响数据写入消费的完整性。

日志配置为 DEBUG 级别时异常

以下异常在日志配置为 DEBUG 级别会出现，客户端会自动处理。

异常	描述	分析与说明
OffsetOutOfRangeException	消费者拉取消息时传入的 Offset 超出范围。	如果客户端设置了 Offset 重置策略（earliest 或 latest），则客户端会根据策略进行 Offset 重置，否则需要用户程序处理这个异常
GroupLoadInProgressException	ConsumerGroup 对应的 Coordinator正在加载。	服务器端升级时可能短暂出现，客户端会自动重试。
GroupCoordinatorNotAvailableException	Coordinator 不可用。	服务器端升级时可能短暂出现，客户端会自动重试。
NotCoordinatorForGroupException	当前节点不是该 ConsumerGroup的 Coordinator，Coordinator 迁移到别的节点。	服务器端升级时可能短暂出现，客户端会自动重试。
IllegalGenerationException	ConsumerGroup 的 generation不合法。	可能心跳超时或有新消费者加入，Consumer 会自动重新尝试加入 ConsumerGroup。
RebalanceInProgressException	ConsumerGroup 正在进行 rebalance。	可能心跳超时或有新消费者加入，Consumer 会自动重新尝试加入 ConsumerGroup。

客户端生产消息时延高

最近更新时间：2025-03-17 14:34:24

问题现象

客户端现象表现为 Producer 消息发送时延增加：

- 消息写入速度变慢，发送时延增大。
- CPU 使用率过高。

排查步骤

步骤1：限流检查

检查单个 Topic 是否被限流：如果限流值，会导致消息发送速率受限。
优化建议：增加 Topic 限流值（基于实际业务需求）。

检查实例是否限流。如果出现限流，需要增加带宽。

步骤2：集群性能检查

检查集群 CPU 负载，如果是集群整体负载高，直接导致发送时延升高。
优化方案：扩容

步骤3：客户端参数优化

合理设置批量参数，减少碎片化请求，提升发送性能。 小批次（小 Batch）会导致客户端频繁发起请求，增加服务端排队压力，进而推高延迟和 CPU 消耗。
优化建议：

- 合理设置 acks \batch.size 和 linger.ms，结合业务需求调整参数，推荐值：

```
acks=1
batch.size=16384
linger.ms=1000
```

详细解释如下：

1. ack 参数调整

Acks 参数控制了 Producer 发送消息后等待的确认机制：

- acks=0：无需服务端确认，性能较高，但丢数据风险大。
- acks=1：主节点写成功即返回确认，性能中等，适合大多数场景。
- acks=all 或 -1：主节点和副本均写成功后返回确认，数据可靠性高，但性能较差。

优化建议：

- 为提升发送性能，建议设置 `acks=1` 。

2. **调整批次发送参数**

批量发送可以减少请求次数，提高网络吞吐量：

- batch.size：每个分区缓存的消息总大小（单位：字节）。当达到设置值时，触发批量发送。
- linger.ms：消息在缓存中停留的最长时间，超过该值后会立即发送，即使未达到 `batch.size` 。

推荐配置：

- `batch.size=16384` （默认值：16KB）。
- `linger.ms=1000` （默认值：0，即不等待直接发送）。

3. **调整缓存大小**

buffer.memory：控制缓存消息的总体大小。缓存超限时会强制触发发送，忽略 `batch.size` 和 `linger.ms` 。

- 默认值为 32MB，对于单个 Producer 而言，可以保证足够的性能。
- 建议配置： `buffer.memory ≥ batch.size * 分区数 * 2` 。

> ⚠ **注意：**
> 如果在同一 JVM 中启动多个 Producer，请谨慎设置 `buffer.memory` ，避免 OOM 问题。

关键问题

1. **CPU 高负载的原因**

   Kafka CPU 的高负载通常来自以下几个方面：

   - 生产和消费请求处理：
     - 高吞吐量的 Producer 和 Consumer 会占用大量 CPU
   - 数据复制：
     - 副本同步的过程需要进行数据复制操作
   - 高频控制请求：
     - 例如 Offset 管理、Metadata 查询等高频操作，会占用 Broker 的 CPU 资源。

2. **如何处理集群高负载**

   集群高负载的常见表现包括：

   - 消息发送延迟增加。
   - 请求队列深度过大，持久维持在高位。

- Broker 的 CPU 或磁盘 I/O 持续高负载。

解决方法：

2.1 **扩容：**

- 增加 Broker 节点数，平摊分区负载，减轻单台服务器压力。
- 优化分区分布，确保分区均匀分布在所有 Broker 上。

2.2 **限流：**

- 设置合理的生产限流值，避免生产端流量暴增导致 Broker 超载。

3. **集群高负载情况，如何优化客户端实现低延迟**

当 Kafka 集群无法扩容或负载无法有效降低时，可以优化客户端配置以尽量获取低延迟。

参考 客户端参数优化

- 合理设置 `acks` \ `batch.size` 和 `linger.ms`，结合业务需求调整参数，推荐值：

```
acks=1
batch.size=16384
linger.ms=1000
```

总结

通过以上排查步骤和参数优化方法，可以有效解决 Kafka 集群负载高和生产消息时延高的问题。

- 对于集群负载高的情况，扩容是最直接的解决方案；
- 对于客户端参数优化，调整批次发送和缓存大小可以显著提高性能。

若问题仍未解决，可联系 在线客服 。

客户端生产消息进入到阻塞状态

最近更新时间：2024-10-11 14:29:52

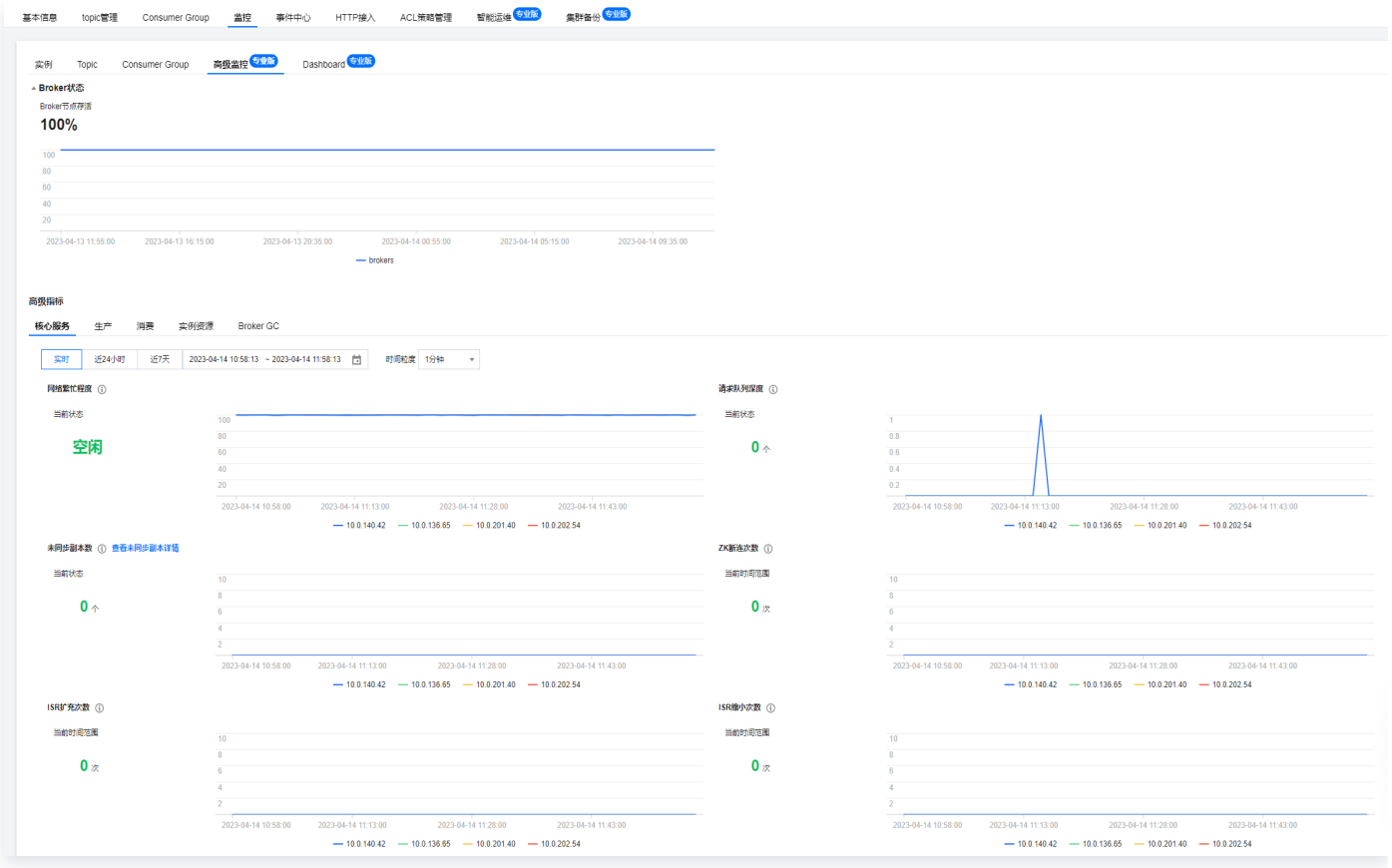
问题概述

客户端生产消息进入堵塞状态，核心原因是消息发送不出去，或者发送的速度小于生产的速度。

- 如果是发送不出去，有 time out 的提示，可以先使用命令行进行生产消费，查看集群基本性能。参见 命令行生产消费 。
- 如果是发送的速度小于生产速度，有三种原因：
  - 生产者的实例太少，即单个生产者的生产性能是有上限的，如果生产者的数量太少，流量太大，可能会导致发送堵塞。
  - 流量太大，Topic 的分区数太少，导致写入并行度不够。
  - 服务的质量有问题，例如网络质量下降，Broker 负载升高，客户端负载升高（例如客户端发生了 GC）会导致客户端发送到服务端的整体耗时上升，导致生产速率下降。从而导致客户端本地的 buffer 堆积，出现阻塞。

可能原因

1. 如果是专业版实例，可以在控制台查看高级监控，观察服务端的整体负载情况，如请求队列深度，生产消费的服务端耗时等。来确认服务端是否有性能问题。如果是标准版实例，可以 提交工单 查看这些指标。



2. 排查客户端负载，如本地机器的 CPU，内存情况（如果是 Java 客户端，重点关注 GC 情况）。
3. 如果是偶尔出现阻塞状况，需要排查本地网络是否有波动。特别是容器网络环境下，需要着重关注。
4. 分析生产者的数量是否过少，可以从单机的流量来分析。如果单机吞吐的流量较大，而生产者又是单线程发送，则需要关注。

解决方法

可以通过以下方法尝试解决问题：

1. 当生产消息的速度比 Sender 线程发送到 Broker 速度快，导致 buffer.memory 配置的内存用完时会阻塞生产者 send 操作，该参数设置最大的阻塞时间。如果需要更大的send buffer，可以通过调大 buffer.memory，buffer.memory 的默认值是 32MB。

```
# 最大阻塞时间
max.block.ms=60000
# 配置生产者用来缓存消息等待发送到 Broker 的内存。用户要根据生产者所在进程的内存总大小调节
buffer.memory=33554432
```

2. 如果 Topic 的流量较大，客户端发送的 Produce 实例较少，可以多起几个 Produce 实例来生产。例如：

```
KafkaProducer<byte[],byte[]> producer = new KafkaProducer<>(props);
```

3. 扩容集群的分区数。
4. 提交工单 申请平台协助处理。

客户端消费不到消息

最近更新时间：2024-10-11 14:29:52

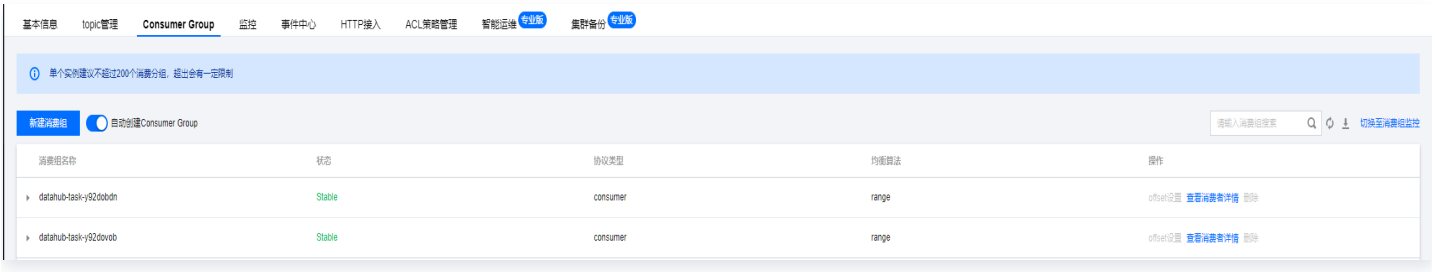
问题概述

消费端拉取不到消息。

可能原因

确认消费者组是否有堆积。如果没有堆积则会在 fetch.max.wait 时间后，返回空消息。该参数是消费者客户端配置的参数，默认是500ms，配置项如下：

```
# Fetch 请求等待时间
fetch.max.wait.ms=500
```



解决方法

- 如果消息有堆积，拉取到的消息却为空，建议检查客户端 SDK 的版本，如果 SDK 版本较老，建议升级 SDK 版本，参见 SDK 文档 。
- 您也可以联系 在线客服 处理。

Sarama 客户端异常

最近更新时间：2025-07-02 15:03:54

问题概述

Sarama 是一个 Golang 编写的 Kafka 客户端，具有较高的消息吞吐性能。
当因为性能达到瓶颈，主动扩容 CKafka 分区后，Sarama 客户端可能会无法感知分区的 rebalance，导致新分区的信息无法被正常生产消费。

可能原因

CKafka 由于各种原因对分区进行 rebalance 后，Sarama 需要大约 10 分钟时间间隔感知分区变动，并拉取当前 topic 的 metadata，解析最新的分区数据。由于拉取时间较久，有时会被用户视作拉取失败。

> **说明：**
> 默认情况下，Sarama 的配置 `sarama.Metadata.RefreshFrequency` 为 10分钟，因此最差需要 20 分钟才能感知到分区更新。 如果希望加速分区更新感知速度，将该配置改小即可。如设置为 10s，则最迟 20s 内就能感知到。

除此之外，在 CKafka 团队长期维护使用中，也发现偶尔会出现 Sarama 客户端拉取最新分区信息失败，导致消息堆积并随机抛出异常的现象。
该场景已经在 Sarama 社区多次提出并且加以关注修复，在迄今为止的最新版本错误出现频率降低，但问题依旧存在。

解决方法

如果用户对 rebalance 现象敏感，并且使用 Golang 技术栈，建议尽快迁移使用 Confluent 维护的客户端 Confluent-Kafka-go 。

常用 Golang 客户端对比

Golang 客户端	优点	局限性
Sarama	• 社区活跃度高。Sarama 项目使用者较多，在社区提出问题得到解答与修复的时间较快。 • 性能较高。Sarama 采用原生 Golang 语言编写，对于异步以及高并发操作支持度较好。	稳定性一般。Sarama 在扩容分区并 rebalance 后可能会有未知错误。
Confluent-Kafka-go（推	• 非常稳定。由于客户端实际上是对于 librdkafka C++ 库的一层封装，	• 增加编译复杂度。由于导入 C++ 库，Golang 编译器需要引入额外

荐）	而 librdkafka 库已经在多种语言上运行多年，能够提供足够的可靠性。 • 性能较高。由于底层使用 C++ 具体实现，所需资源较少，运算速度快。	编译配置，增加了编译依赖，提高编译复杂度。 • 除此之外，由于不同编译环境 C++ 库实现逻辑不同，引入 librdkafka 库可能会对 Golang 项目交叉编译造成影响。
kafka-go	• 接口完善。kafka-go 不仅提供顶层接口，同时也暴露底层接口。用户可以更为灵活的调用配置 kafka 客户端。 • 操作简单。kafka-go 在进行基础的生产消费所需代码较少，具有较多的缺省配置。	与前两款客户端相比性能较差，可能无法满足大规模并发需求。

消息相关故障
消费进度未及时同步到服务端

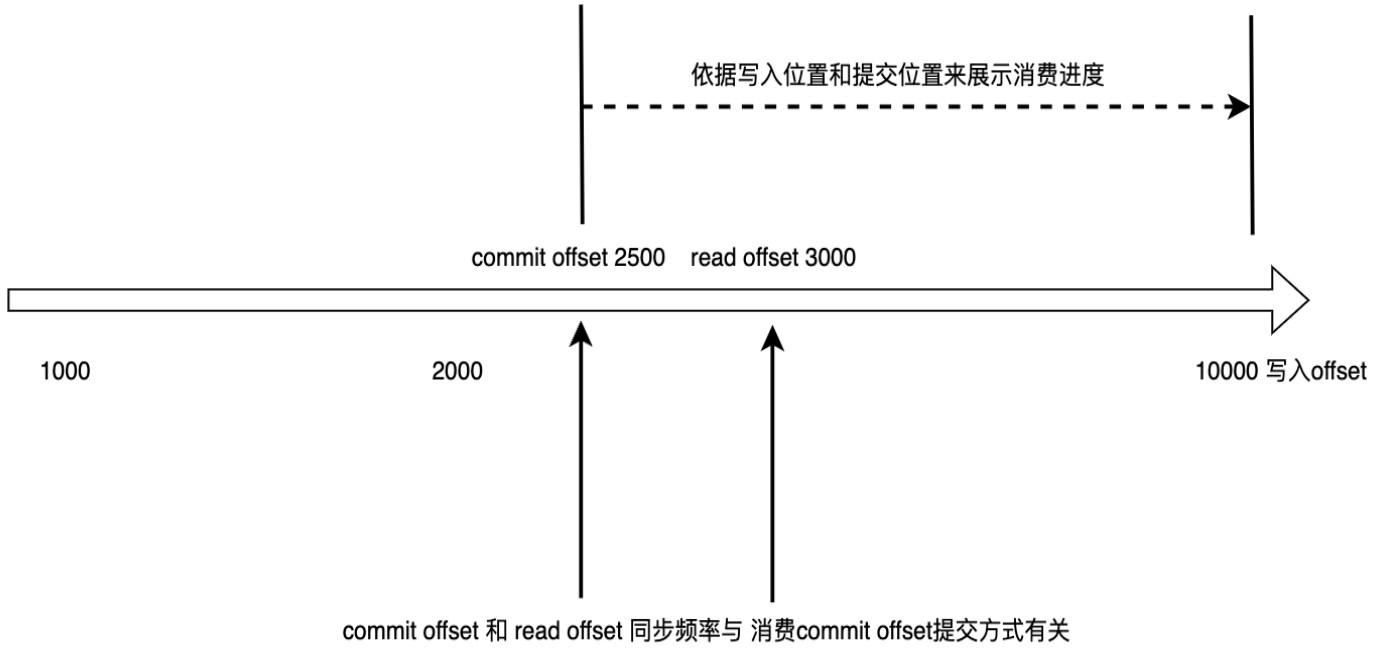
最近更新时间：2025-03-17 14:34:24

问题现象

CKafka 消费进度未及时更新服务端，无法通过控制台看到消费速度。

前置储备

Ckafka 集群展示消费进度，是以客户端消费位移 offset 的提交频率来评估消费速度和消费进度，故遇到此类问题优先排查客户端消费提交 offset 相关情况。

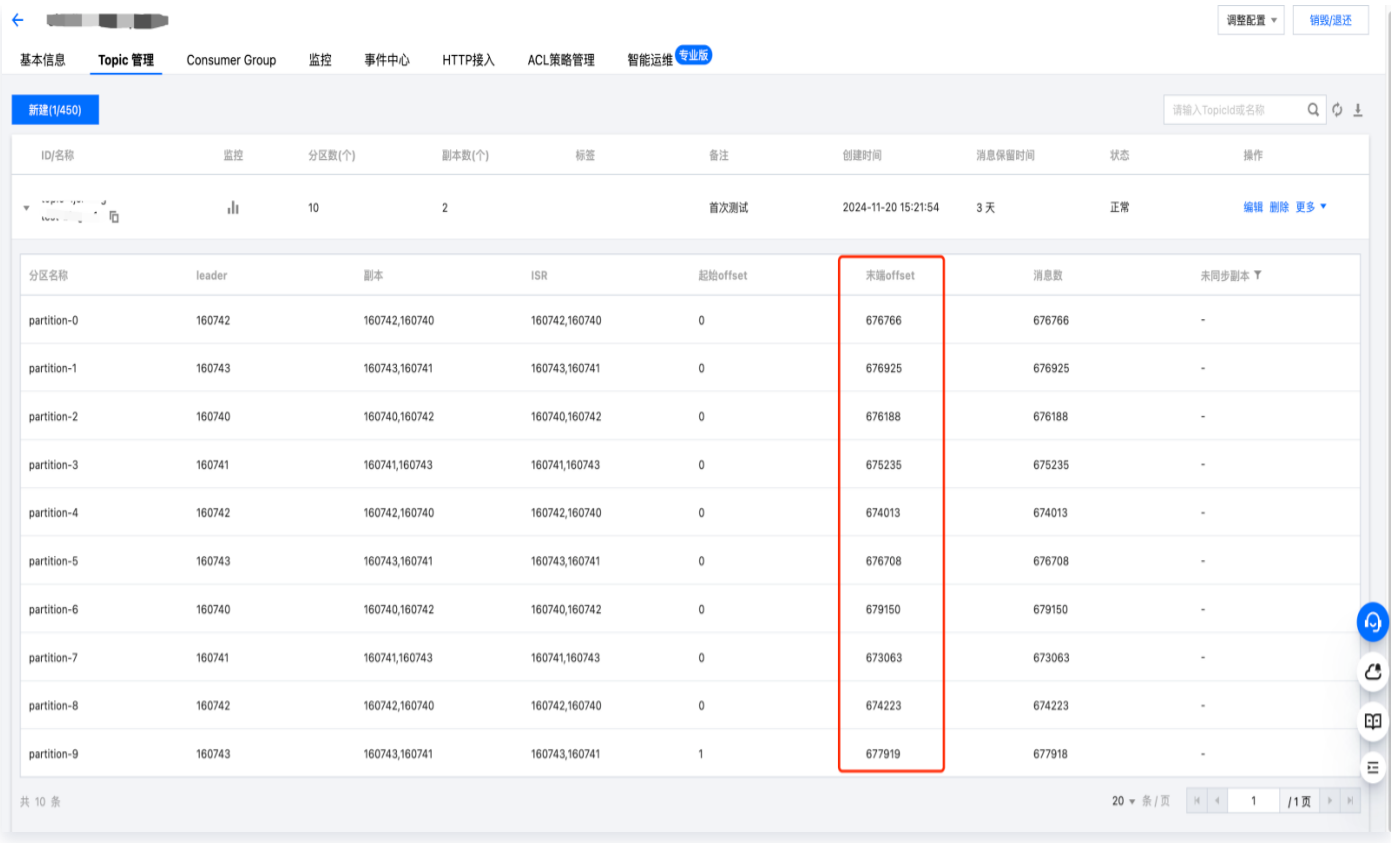


排查指引

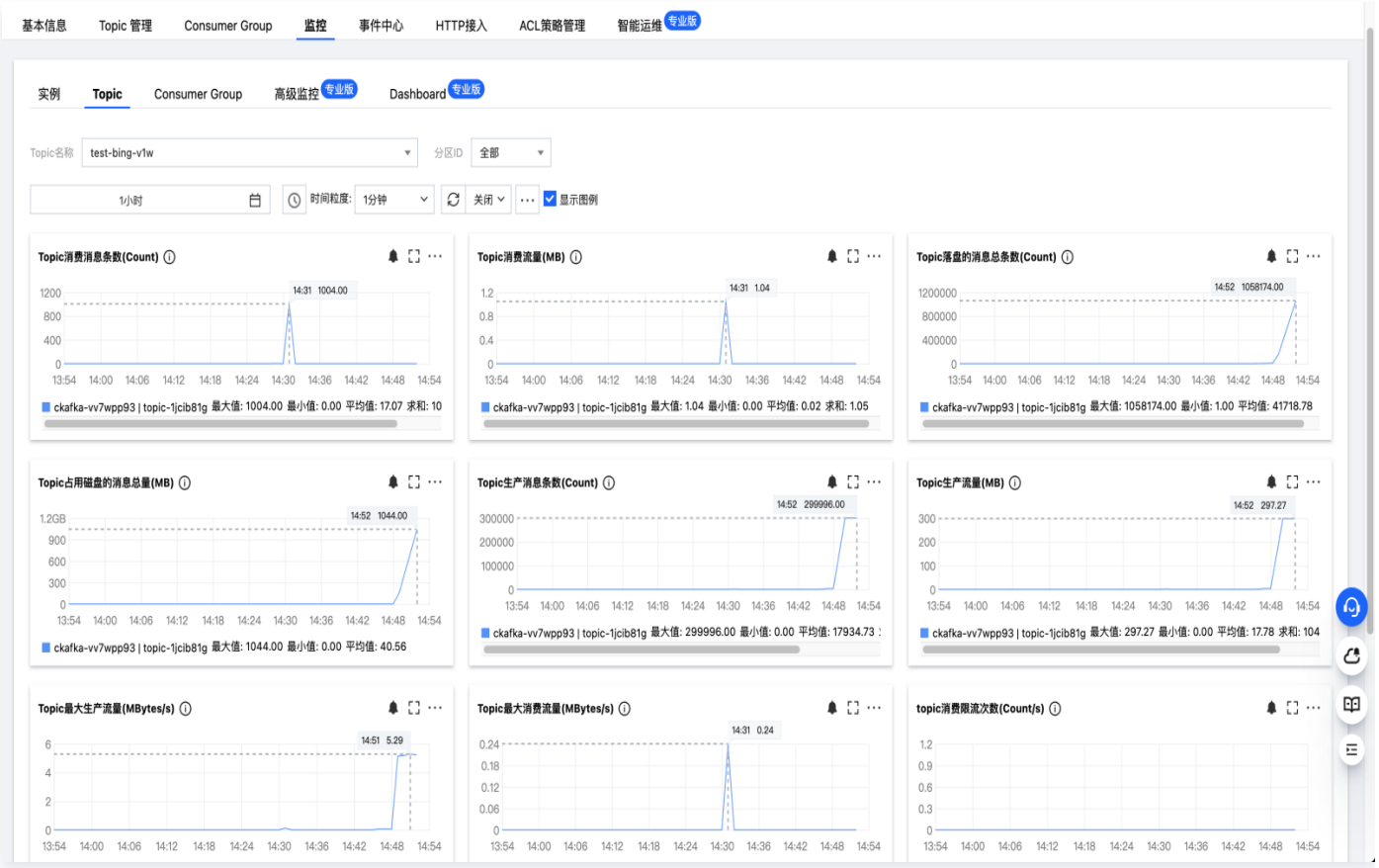
步骤1：检查是否有生产数据写入

如果发现消费没有出现堆积和消费速度，可检查 Topic 以及生产监控指标，查看是否有数据写入。

1. 进入 控制台，单击**实例列表** > **ID/名称** > **Topic 管理**，查看对应 Topic 相关分区末端 offset 是否会更新。



2. 进入 控制台，单击**实例列表 > ID/名称 > 监控 > Topic**，查看对应 Topic 相应生产流量、Topic 占用磁盘的消息总量等指标是否大于 0。



如果 Topic 无实时数据写入，有下述两种情况：

- 新建消费组，配置 `auto.offset.reset=latest` 行为，则会从最新开始消费，由于没有实时数据写入，不会产生 `commit offset` 行为；
- Topic 数据过了保留时间，配置 `auto.offset.reset=earliest` 行为，则会从最头开始消费，由于数据已经 `retention`，消费不到数据，也不会产生 `commit offset` 行为。

上述情况不产生 `commit offset` 行为，故而，服务端不会更新消费进度。

步骤2：检查客户端配置

对于 Kafka 原生 Client，需要检查 Consumer 配置，是否开启自动提交 `offset：enable.auto.commit` 。

原生 Kafka Java Client 举例

一种直接检查客户端配置，还有一种方式可以从程序的日志里面搜索 ConsumerConfig，如下图所示。该日志只在 KafkaConsumer 初始化时打印，如果日志保留时间较短可能搜索不到。

```
2024-11-21 10:33:25,668 INFO  org.apache.kafka.clients.consumer.ConsumerConfig          [] - ConsumerConfig values:
    allow.auto.create.topics = true
    auto.commit.interval.ms = 5000
    auto.offset.reset = latest
    bootstrap.servers = [
    check.crcs = true
    client.dns.lookup = use_all_dns_ips
    client.id =
    client.rack =
    connections.max.idle.ms = 540000
    default.api.timeout.ms = 60000
    enable.auto.commit = true
    exclude.internal.topics = true
    fetch.max.bytes = 52428800
```

- 依据客户端消费特点，如果可以开启自动提交 offset，则可以打开该参数，客户端消费进度将更新到服务端；
- 如果需要手动管理 offset 的提交，例如对重复消费或丢数有更严格要求，`enable.auto.commit` 被关闭的情况下，需要检查客户端程序，是否在消费过程中，同步或者异步执行 offset 的提交动作，例如 consumer.commitSync()

```
while (true) {
            ConsumerRecords<String, String> consumerRecords =
consumer.poll(500);
            for (ConsumerRecord record : consumerRecords) {
                System.out.printf("收到消息: partition:%d, offset:%d,
time:%d, key:%s, value:%s\n", record.partition(), record.offset(),
record.timestamp(), record.key(), record.value());
            }
            consumer.commitSync();
        }
```

原生 Kafka Client 经过上述排查调整后，再次观察客户端消费进度是否更新到服务端。

针对 Kafka 流式场景，Flink 出现频率相对较多

对于 Flink 消费客户端，其底层与 Kafka 的访问集成至 Flink的Kafka Source 接口，并与 Flink 框架容错机制相结合，例如 checkpoint 和 savepoint 机制。在这里就会面对两种情况：Kafka Client 底层 enable.auto.commit 处理和 Flink checkpoint 对 offset 的处理。

- Kafka Source 在 checkpoint 完成时提交当前的消费offset ，以保证 Flink 的 checkpoint 状态和 Kafka broker 上的提交位点一致。
- 如果未开启 checkpoint，Kafka Source 依赖于 Kafka consumer 内部的位点定时自动提交逻辑，自动提交功能由 enable.auto.commit 和 auto.commit.interval.ms 两个 Kafka consumer 配置项进行配置。需要注意的是，如果用户在使用 Flink Kafka Source 时没有主动在配置中指定 enable.auto.commit，Flink框架将会覆盖 enable.auto.commit=false，即关闭自动提交 offset，如果 Flink 未开启 checkpoint，则需要用户手动开启该参数。

消费数据异常

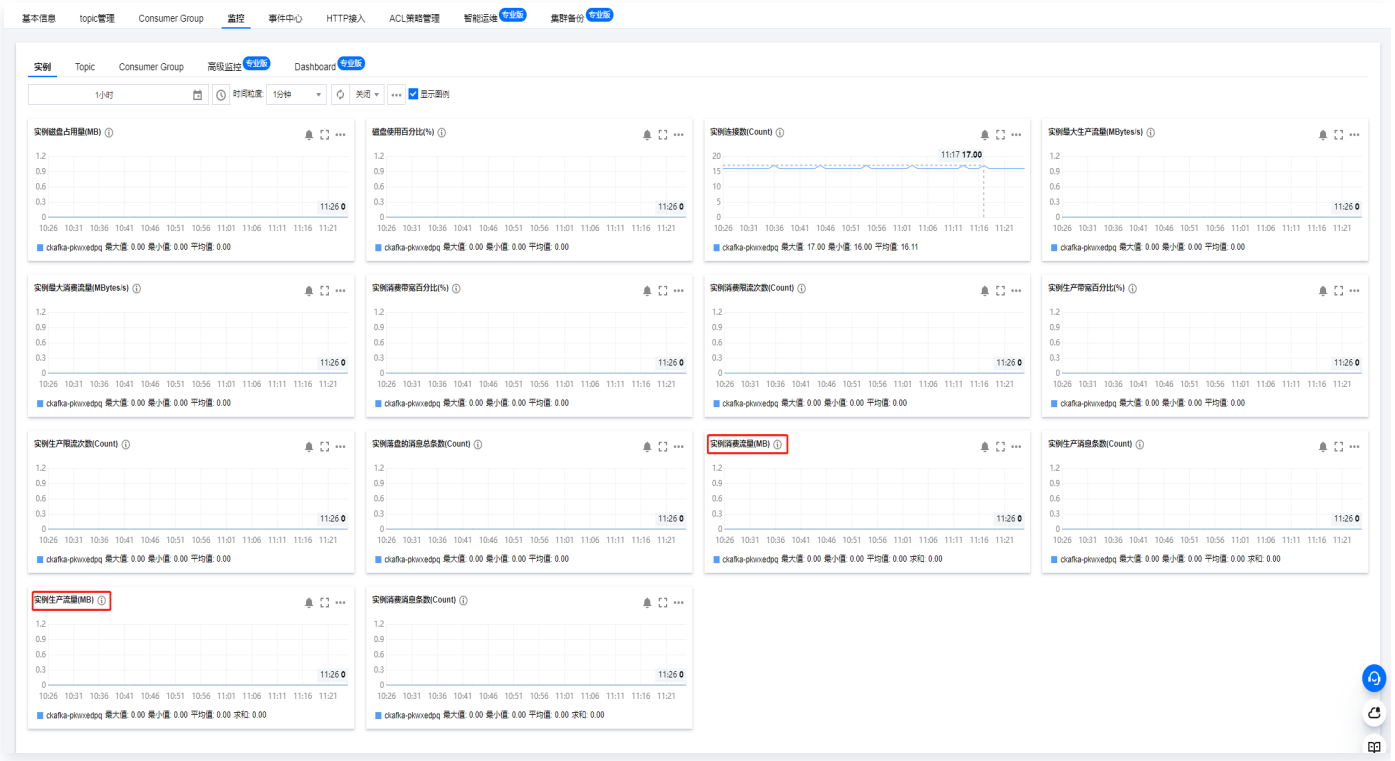
最近更新时间：2024-10-11 14:29:52

问题概述

消费数据异常。

排查思路

- 在 CKafka 控制台监控页面查看流量监控情况，观察是否存在波峰，如果存在则通过升级实例大小解决。



- 查看消费分组是否超过数量。
- 如果因为网络频繁 rebalance，建议调整客户端超时时间。
- 是否拉取过期的 offset，消息过期会被删除，如果用过期 offset 拉取会失败。

过期消息没有被及时删除

最近更新时间：2024-10-11 10:50:52

问题概述

过期消息没有及时被删除。

原因分析

Kafka 的消息删除机制会导致某些业务场景出现过期消息没有及时删除的情况，如果对机制不了解容易产生疑惑，例如：分区0和分区7的消息时间戳存在明显差距，分区0的过期消息没有被及时删除。

Kafka 消息删除机制

Kafka 数据存储是以 Topic、分区、数据段三个维度实际落盘存储的，消息数据删除的条件如下：

- 消息数据根据保留时间进行删除，删除是以数据段为单位的。
- 每个数据段当前是设置为1GB大小，达到1GB后滚动生成新的数据段。
- 数据段内的所有消息都过期才会删除该数据段。
- 如果数据段内有一行消息在保留时间内，即例如段文件的最后一行是在保留时间内，这个段文件就不会被删除。

由于某些原因导致消息写入有倾斜，数据写入集中在某些分区，例如分区7，某些分区数据很少，例如分区0。此时分区0的数据段大小未达到1GB，没发生滚动，但整个段内有数据在保留时间内，所以分区0中的消息就不会被删除。

消费速度缓慢

最近更新时间：2024-12-03 11:12:32

问题概述

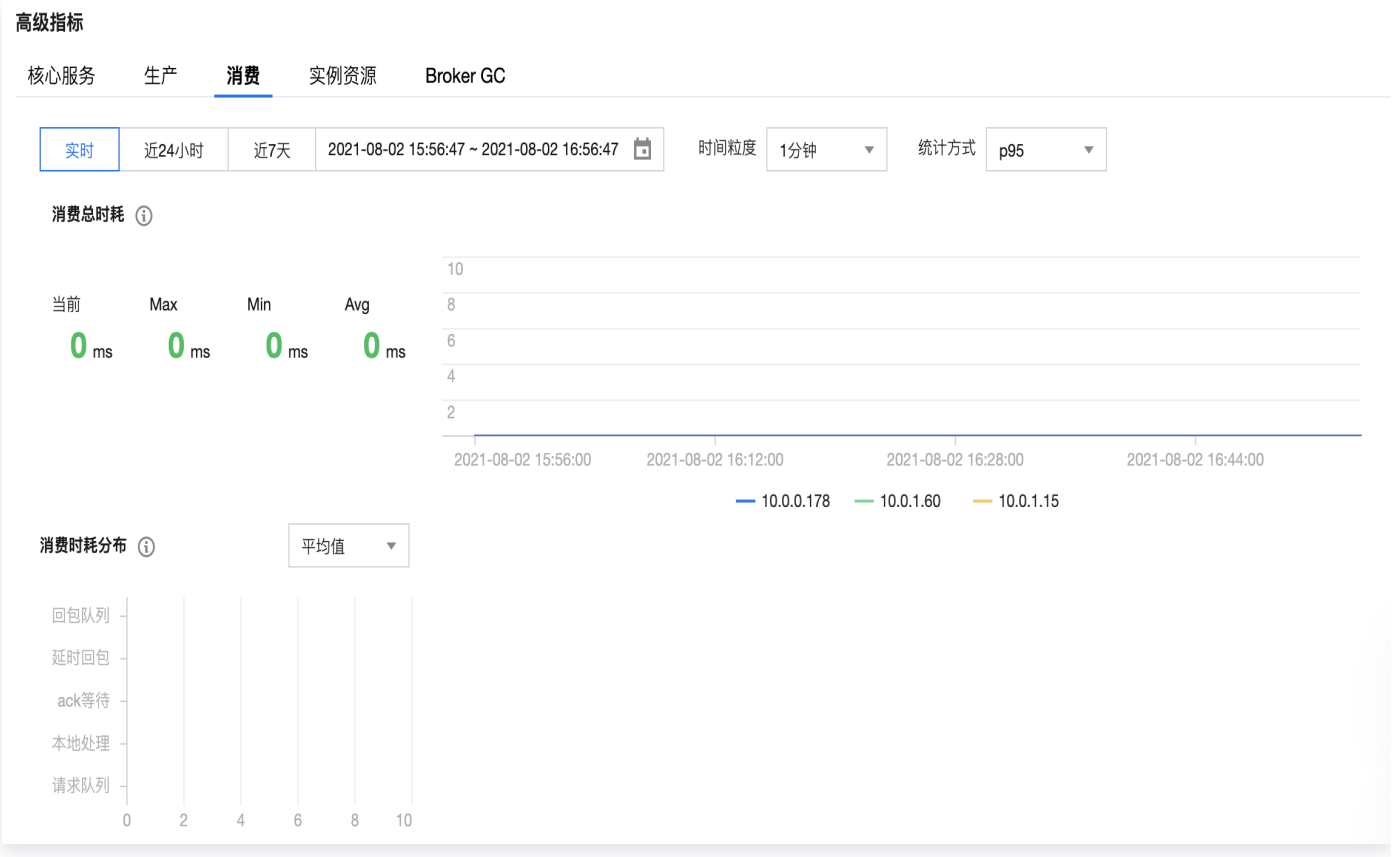
消费端消费消息速度缓慢。

可能原因

- 服务端负载较高
- 限流问题
- 客户端负载
- 消费端处理能力问题
- 网络问题

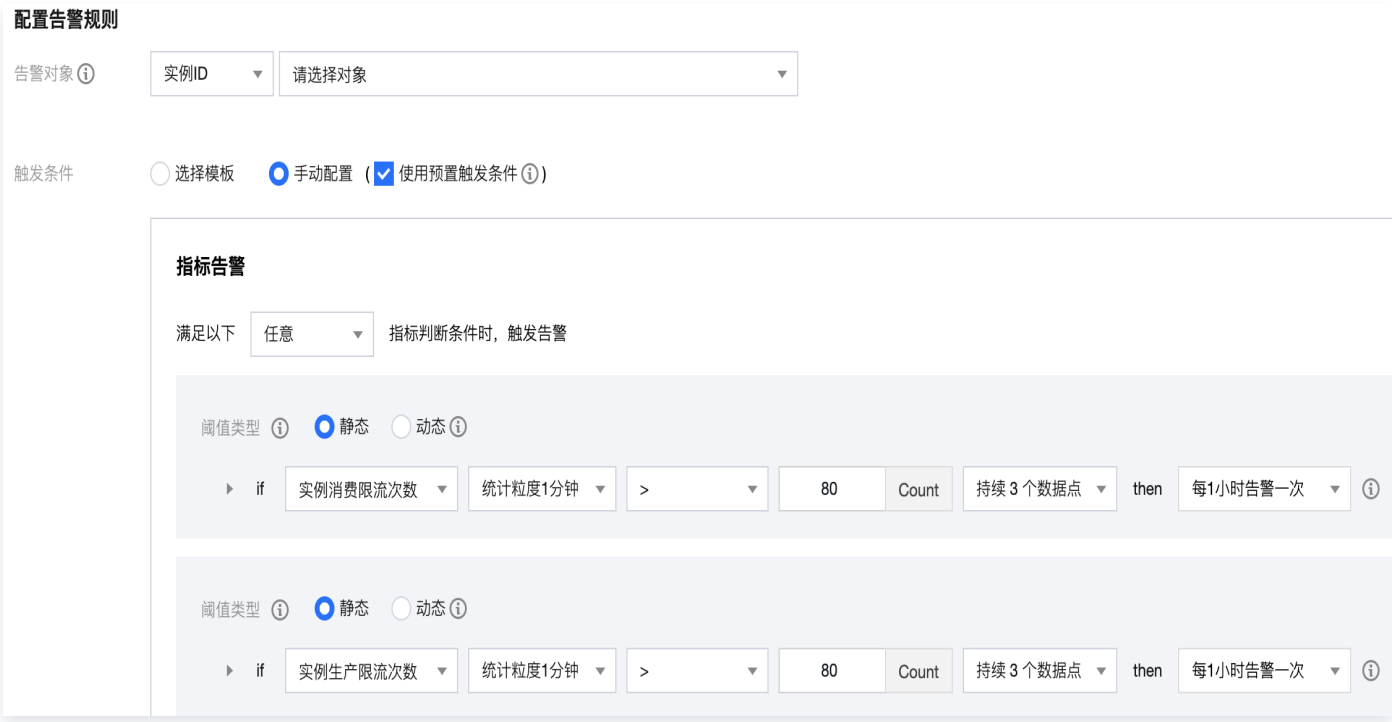
解决方法

- **服务端负载较高**
- 如果想确认是否是服务端问题，可以在控制台查看高级监控里面的消费耗时，耗时信息表示服务端处理请求的耗时，如果服务端负载有问题，可以看到统计各阶段耗时较高，如下图：

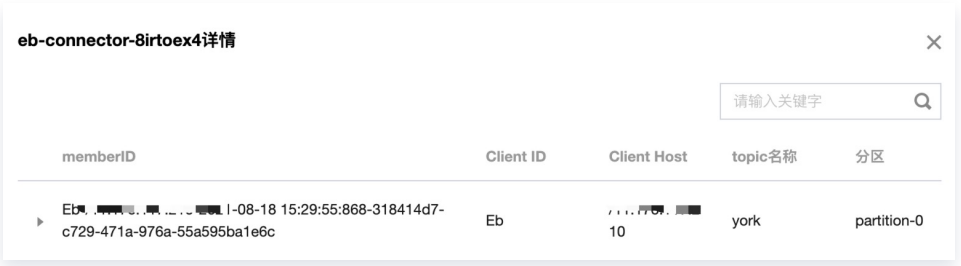


- **限流问题**

  如果想确认是否是限流问题引起的，可以配置带宽超限告警。检查 **监控** > **实例** 是否已经达到实例的带宽峰值。如果已经达到带宽峰值，您需要升级实例的带宽峰值。关于如何升级实例配置，请参阅 升配实例 。



- **客户端负载**
- 如果服务端没有性能问题，大概率是客户端消费能力不足。首先看一下分区和消费者的对应关系。如果一个消费者消费了太多分区，建议增加消费者的数量。尽量让一个消费者消费一个分区，如下图查看消费者和分区的对应关系：



- **消费端处理能力问题**

  如果消费者和分区的分配关系是正常的，那可以在控制台扩容分区提高数据消费的并行度。控制台扩容分区是即时无损的扩容的，不会影响您的业务。扩容分区如下图：

- **网络问题**

  排查一下客户端的负载情况。例如客户端机器的 CPU、内存、网卡等指标。如果是 Java 的进程，则着重关注 GC 和堆内存的使用情况。

- **消费链路问题**

  排查消息消费链路下游方面，是否存在其他原因导致消费速度慢。例如消费消息需要有一些同步操作完成后，才向服务端提交 offset。下游常见的异常操作有：

  - 频繁打印日志。
  - 写入数据库较慢。
  - 业务逻辑较为复杂，导致处理消息较慢。

  如果短时间不能快速解决，建议增加 Topic 分区数，并扩容消费者实例数量，以迅速增加消费消息速度。

出现消息堆积的警告

最近更新时间：2024-11-21 15:16:42

问题概述

出现消息堆积的警告。

常见原因

- 客户端没有消费。
- 客户端消费速度较慢。
- 服务端异常。

排查思路

- **首先，判断是服务端问题，还是客户端问题。**

  1.1 通过控制台，查看实例级别监控，判断实例是否已经达到最高带宽，是否有产生限流；

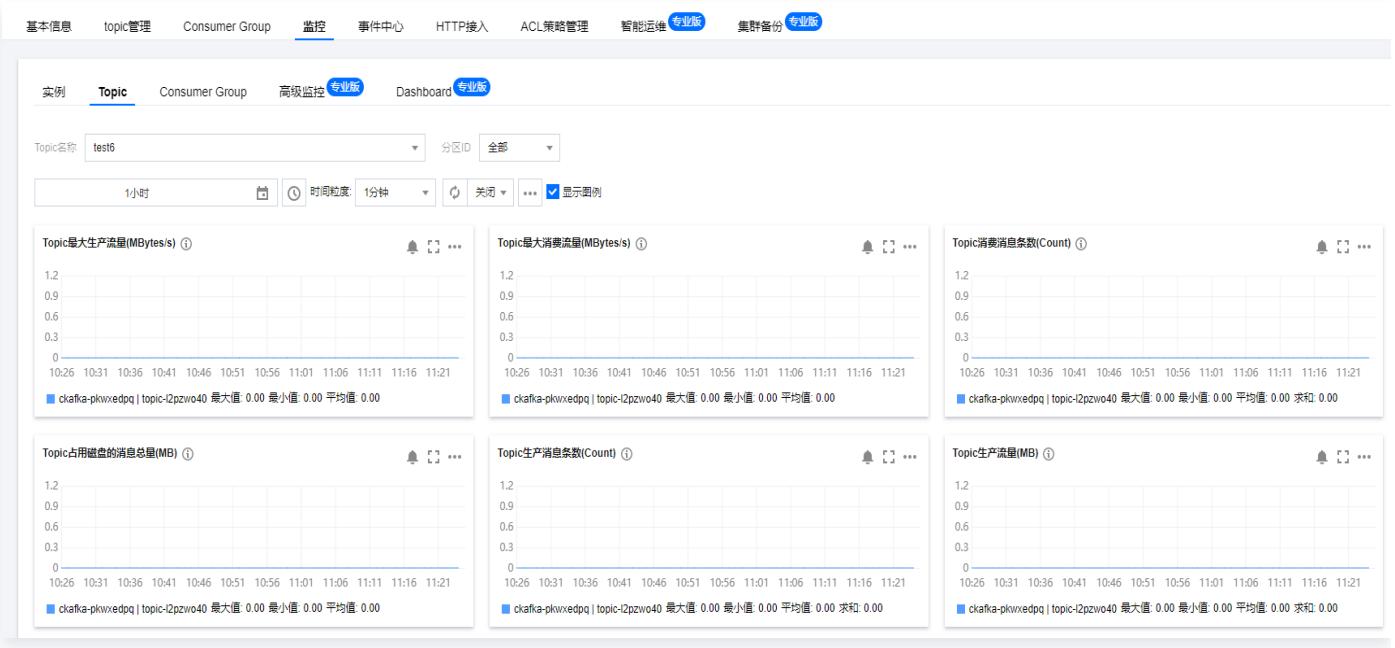
  1.2 如果您是专业版，可以切换到「高级监控」页：

  - 查看 Broker 节点存活、网络空闲；
  - 查看消费 Sheet 页下，消费总耗时、消费耗时分布，详细指标说明请参考高级监控（专业版）。

  1.3 排除上述指标异常后，可以确定排查方向转向客户端。

- **客户端没有消费**

  可以通过查看分区的消费速度来确认，是否有进行消费，如下图：



- **客户端消费速度较慢**

  请参见 消费端消费消息速度缓慢 。

推荐设置

开源 Kafka 支持消息中设置一个时间戳字段和时间戳类型，目前支持的时间戳类型有两种： CreateTime 和 LogAppendTime。

- CreateTime 表示客户端本地的时间，由于客户端的时间可能和服务器时间存在偏差，请检查写入的时间是否是正确的时间。如果时间和当前北京时间相差较远，导致 CKafka 服务无法按照正常消息保存时间对数据进行及时过期删除，因此可能会存在消息异常堆积 。
- LogAppendTime 表示消息生产到 CKafka 服务的时间，时间为 CKafka 服务器的时间，建议用户选用 LogAppendTime。

压测服务端性能

如果对服务端性能有疑问，也可以执行如下压测命令来排除是否是服务端存在问题。命令如下：

- 生产测试命令示例：

```
bin/kafka-producer-perf-test.sh
--topic test
--num-records 123
--record-size 1000
--producer-props bootstrap.servers= ckafka vip : port
--throughput 20000
```

- 消费测试命令示例：

```
bin/kafka-consumer-perf-test.sh
--topic test
--new-consumer
--fetch-size 10000
--messages 1000
--broker-list bootstrap.servers=ckafka vip : port
```

详情请参见 对 CKafka 进行生产和消费压力测试 。

生产一段时间后发现持续性错误

最近更新时间：2024-10-14 17:10:22

问题概述

生产一段时间后发现持续性错误。

排查思路

- 查看是否流量流控，在监控页面上查询是否有波峰，升级实例大小解决。
- 查看是否容量流控，在监控页面上查询实例的磁盘容量，升级实例大小或者修改数据保存时间。